SIMD instruction set extensions for multimedia

SIMD multimedia extensions started with the simple observation that many media applications operate on narrower data types than the 32-bit processors were optimised for. For example, many graphics systems used 8 bits to represent colours plus another 8 bits to represent transparency, for a grand total of 16 bits.

By partitioning the carry chains within a 256-bit adder, for instance, a processor could perform simultaneous operations on short vectors of thirty-two 8-bit operands, sixteen 16-bit operands, eight 32-bit operands or four 64-bit operands, parallelising execution. The additional cost of such partitioned adders is small.

Differences from vector architectures

Like vector instructions, a SIMD instruction specifies the same operation on vectors of data. However, unlike vector machines with large register files (such as the VMIPS vector register), SIMD instructions tend to specify fewer operands and hence use smaller register files.

In contrast to vector architectures, which offer an elegant instruction set that is intended to be the target of a vectorising compiler, SIMD extensions have **three major omissions**:

1. Multimedia SIMD extensions **fix the number of data operands in the opcode**, which has led to the addition of hundreds of instructions in the Multimedia Extensions (MMX), Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX) extensions of the x86 architecture.
2. Multimedia SIMD does not offer the more sophisticated addressing modes of vector architectures, namely strided accesses and gather-scatter accesses[1]
3. Multimedia SIMD usually does not offer the mask registers to support conditional execution of elements[1-1]

Instruction set extensions for the x86 architecture

For the x86 architecture, the Multimedia Extensions (MMX) instructions added in 1996 repurposed the 64-bit floating-point registers, so the basic instructions could perform eight 8-bit operations or four 16-bit operations simultaneously. These were joined by parallel `MAX` and `MIN` operations, a wide variety of masking and conditional instructions, operations typically found in digital signal processors and ad hoc instructions that were believed to be useful in important media libraries.

The Streaming SIMD Extensions (SSE) successor in 1999 added separate registers that were 128 bits wide, so now instructions could simultaneously perform sixteen 8-bit operations, eight 16-bit operations or four 32-bit operations. It also performed parallel single-precision floating-point arithmetic. Since SSE had separate registers, it needed separate data transfer instructions. Intel continued expanding the SSE instructions until SSE4 in 2007. With each generation, they also added ad hoc instructions in order to accelerate specific multimedia functions.

The Advanced Vector Extensions (AVX), added in 2010, doubles the width of the registers again to 256 bits.

Advantages and drawbacks

In general, the goal of these extensions has been to accelerate **carefully written libraries** rather than for the compiler to generate them. But given this fact, why were multimedia SIMD extensions so popular?

First of all, they cost little to add to the standard arithmetic unit and were easy to implement.

Second, they require little extra state compared to vector architectures, which is concerning for context switch time.

Third, a lot of memory bandwidth is necessary to support a vector architecture, which many computers don't have.

Fourth, SIMD doesn't have to deal with problems in virtual memory when a single instruction that can generate 64 memory accesses can get a page fault in the middle of the vector.

Another advantage of short, fixed-length "vectors" of SIMD extensions is that it's easy to introduce instructions that can help with new media standards.

Finally, where was concern about how well vector architectures could work with caches.

---

1. These properties of vector architectures weren't covered by the course, hence there are no notes related to them. However, they are explained in great detail on the book I studied from, which you can find in the index page of the notes of this course. ↩ ↩